

NAVAL POSTGRADUATE SCHOOL

Monterey, California



19990921 070

THESIS

ANALYSIS OF THE NECKLACE ALGORITHM
AND ITS APPLICATIONS

by

Douglas M. Matty

June 1999

Thesis Advisor:
Second Reader:

Harold M. Fredricksen
Craig W. Rasmussen

Approved for public release; distribution is unlimited.

DTIC QUALITY INSPECTED 4

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY (Leave blank)

2. REPORT DATE
June 1999

3. REPORT TYPE AND DATES COVERED
Master's Thesis

4. TITLE AND SUBTITLE

ANALYSIS OF THE NECKLACE ALGORITHM AND ITS APPLICATIONS

5. FUNDING NUMBERS

6. AUTHOR(S)

Matty, Douglas M.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING
ORGANIZATION REPORT
NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Department of Mathematics, Naval Postgraduate School, Monterey, CA
93940

10. SPONSORING AGENCY
REPORT NUMBER

11. SUPPLEMENTARY NOTES

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (maximum 200 words)

A commonly studied problem in the field of cryptography is the Discrete Logarithm Problem. This problem is also referred to as the "distance" problem. Basically, one would like to know where a particular binary n-tuple is in a list combining all of them, represented as powers of some primitive element, or equivalently what is the distance between a given pair of n-tuples in a similar representation. A de Bruijn sequence is a well-known periodic binary sequence in which every n-tuple from 0 to 2^n-1 appears. Our goal is to better understand the "prefer-ones" de Bruijn sequence. Ultimately, we wish to understand where each of the binary n-tuples appears in that sequence. Using the Necklace Algorithm, the sequence of n-tuples can be generated. This list has some special properties that allow us to perform the required analysis to locate the n-tuples by an association into classes. We partition the binary n-tuples into necklace classes according to the longest substring of ones appearing on the n-tuple. We then count how many n-tuples appear in the sequence for the first time as members of a necklace class containing no longer strings of ones.

14. SUBJECT TERMS

Cryptography, Combinatorics, Discrete Mathematics, Analysis of Algorithms

15. NUMBER OF
PAGES

53

16. PRICE CODE

17. SECURITY CLASSIFICATION OF
REPORT

Unclassified

18. SECURITY CLASSIFICATION OF
THIS PAGE

Unclassified

19. SECURITY CLASSIFI- CATION
OF ABSTRACT

Unclassified

20. LIMITATION
OF ABSTRACT

UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

**ANALYSIS OF THE NECKLACE ALGORITHM
AND ITS APPLICATIONS**

Douglas M. Matty
Captain, United States Army
B.S., United States Military Academy, 1990

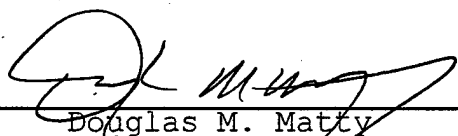
Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN APPLIED MATHEMATICS

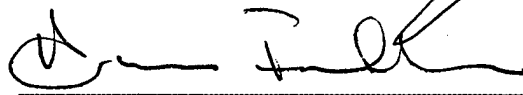
from the

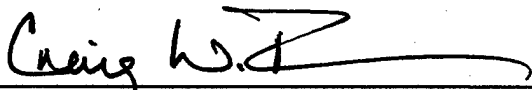
**NAVAL POSTGRADUATE SCHOOL
June 1999**

Author:


Douglas M. Matty

Approved by:


Harold M. Fredricksen, Thesis Advisor


Craig W. Rasmussen, Second Reader


Michael A. Morgan, Chairman
Department of Mathematics

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

A commonly studied problem in the field of cryptography is the Discrete Logarithm Problem. This problem is also referred to as the "distance" problem. Basically, one would like to know where a particular binary n -tuple is in a list combining all of them, represented as powers of some primitive element, or equivalently what is the distance between a given pair of n -tuples in a similar representation. A de Bruijn sequence is a well-known periodic binary sequence in which every n -tuple from 0 to 2^n-1 appears. Our goal is to better understand the "preferences" de Bruijn sequence. Ultimately, we wish to understand where each of the binary n -tuples appears in that sequence. Using the Necklace Algorithm, the sequence of n -tuples can be generated. This list has some special properties that allow us to perform the required analysis to locate the n -tuples by an association into classes. We partition the binary n -tuples into necklace classes according to the longest substring of ones appearing on the n -tuple. We then count how many n -tuples appear in the sequence for the first time as members of a necklace class containing no longer strings of ones.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I. INTRODUCTION.....	1
A. BINARY N-TUPLES.....	1
B. NECKLACE ALGORITHM.....	1
1. Definition of Necklace Algorithm	1
2. Properties of the Necklace Algorithm.....	3
3. Prenecklace Generation.....	5
4. Necklaces.....	10
II. COUNTING THE NECKLACE ALGORITHM.....	13
A. PRENECKLACE COUNT	13
1. Table of Data for Missed n-tuples.....	13
2. Plot of Missing n-tuples Relative to m	18
3. Finding Steady State	21
B. NECKLACE COUNT.....	26
1. Table of Data for Omitted Necklaces and n-tuples.....	26
C. DEFINING STEADY STATE	29
III. CONCLUSIONS AND SUGGESTIONS FOR ADDITIONAL RESEARCH.....	33
APPENDIX A. TABLE OF MISSING N-TUPLES COUNTS.....	37
APPENDIX B. CODE FOR NECKLACE ALGORITHM.....	39
APPENDIX C. TABLE OF LYNDON WORD COUNTED BY PARTITIONS.....	41
LIST OF REFERENCES.....	43
INITIAL DISTRIBUTION LIST.....	45

I. INTRODUCTION

A. BINARY N-TUPLES

The set of binary n -tuples is the set of all strings of 0's and 1's of length n . Usually these are ordered from 00...0 to 11...1. These n -tuples can be found on necklaces defined by cyclic rotation. These necklaces are Lyndon words of length d , where $d|n$. Example for $n = 4$:

(0) (0001) (0011) (01) (0001) (1)

These are the Lyndon words (0) and (1) of length one, (01) of length two, and (0001), (0011) and (0001) of length four.

B. NECKLACE ALGORITHM

1. Definition of Necklace Algorithm

The Necklace Algorithm is an algorithm for generating the necklaces of beads of length n in two colors, i.e., the binary necklaces. Fundamental to the Necklace Algorithm is the operation, $\Theta: V_p^n \rightarrow V_{p+1}^n$, defined as:

$$\Theta(a_1, a_2, \dots, a_n) = (b_1, b_2, \dots, b_n);$$

where $b_i = a_i$ for $i = 1, 2, \dots, j-1$ and j is determined as the largest subscript such that $a_j > 0$ and $a_k = 0$ for all $k > j$. Then $b_j = a_j - 1$, and $b_{j+t} = b_t$ for $t = 1, 2, \dots, n-j$.

Necklace Algorithm (for necklaces of length n).

Step 0. The initial necklace, V_0^n , is $11\dots 1 = 1^n$;

Step 1. Θ -step: Form $\Theta(V_p^n)$ to find V_{p+1}^n ;

Step 2. J -check: The resulting string is the next necklace if and only if $j|n$.

Step 3. If we have not found the last necklace, 0^n , by steps 1 and 2 above, return to step 1. [1]

Figure 1. Necklace Algorithm

Note: If step 2 is omitted, we produce all prenecklaces (defined later). If step 2 is modified such that $j = n$, we produce a list of all Lyndon words of length n .

The following output of the Necklace Algorithm using the three options for the J -check is provided for $n = 4$:

No J -check	$j n$	$j=n$
1111	1111	
1110	1110	1110
1101		
1100	1100	1100
1010	1010	
1001		
1000	1000	1000
0000	0000	

Table 1. Necklace Algorithm Output ($n = 4$)

Thus the Necklace Algorithm produces a listing of the n -tuples appearing from the largest to the smallest and with d of the n -tuples appearing on each Lyndon word of length d .

Additionally, the Lyndon words that form the necklaces can be juxtaposed in order to create the de Bruijn "prefer-one's" sequence as follows: 1·1110·1100·10·1000·0. [2]

2. Properties of the Necklace Algorithm

The Necklace Algorithm output is the list of n -long necklaces, where the largest n -tuple on each necklace represents the necklace and all the necklaces appear in lexicographic order from largest to smallest. Other authors order necklaces and their respective Lyndon words from smallest to largest and represent a necklace by its least representative. The results are equivalent and it is easy to translate between them. Inherent in this list, the n -tuples are partitioned into classes defined according to the length of the initial string of 1's in the n -tuple. We make some clarifying definitions.

Definition 1. An n -tuple of length n has length

parameter l , where $l = \begin{cases} \frac{n}{2} & n \text{ is even} \\ \frac{n-1}{2} & n \text{ is odd} \end{cases}$, and is a member of the

class, m . These parameters also indicate the number of leading 1's as there are $l-m$ leading 1's before the first 0. Each class then has the general form with the respective parameters as follows:

$$\overbrace{1\dots 10}^{l-m} \overbrace{\alpha_{l-m+2} \dots \alpha_n}^{DOF} \text{ such that } l-n \leq m \leq l.$$

Note: The extreme special cases are the n -tuple, 1^n , where $m=l-n$ since there is no zero following the lead string of ones, and 0^n , where $m=l$ since no ones preceding the first zero.

The leading substring is defined as $\overbrace{1\dots 10}^{l-m}$, and is common to all members of a respective m class for a given l . To consider the number of members of a class, we also consider the degrees of freedom (DOF), i.e., those bits that are unspecified by the characteristics of the class. We consider a particular n -tuple class as an example:

$$V = 1110\alpha_5\alpha_6\alpha_7\alpha_8\alpha_9\alpha_{10}\alpha_{11}$$

$$l: \quad 5$$

$$n \text{ (Odd)}: \quad 11 \quad (\text{e.g. } 2l+1 = 2(5)+1 = 11)$$

$$m: \quad 2 \quad (\text{e.g. } l-(l-m) = 5-3 = 2)$$

$$DOF: \quad 7 \quad (\text{e.g. } 2l+1-[(l-m)+1] = 11-[(5-2)+1] = 7)$$

A subtle point is to be noted. The leading 1's in a class m will increase in number with l , and thus with n , while m remains fixed. So an 11-tuple in class 3 has two leading 1's, while a 13-tuple in class 3 has three leading

"1's." The 11-tuple has 8 degrees of freedom while the 13-tuple has 9 degrees of freedom.

The general form of the output of the Necklace Algorithm is refined to represent the output into the respective m classes. The first output in class m appears by applying Θ to the last output of the class $m-1$. The last element of the class $m-1$ is denoted as $[l-m+1]$ and the first element of the class m is then denoted as $\Theta[l-m+1]$, where

$$\Theta[l-m+1] = \overbrace{11 \dots 10}^{l-m} \overbrace{11 \dots 10}^{l-m} \overbrace{11 \dots 10}^{l-m} \dots \overbrace{111 \dots}^{n(\bmod(l-m+1))},$$

and the last output of the Necklace Algorithm for the class, m , referenced as $[l-m]$, has the form:

$$\overbrace{111000 \dots 0}^{l-m}.$$

3. Prenecklace Generation

A prenecklace appearing by step 1 of the Necklace Algorithm is defined as follows:

Definition 2. An n -vector, V^n , is a prenecklace if there exists some k -vector V^k such that $V^{n+k} = f(V^n, V^k)$, where V^{n+k} is a necklace of length $n+k$. (Note: $f()$ denotes the operation of concatenation between V^n and V^k) [3].

Theorem 1. The vector $V^n = v_1 v_2 \dots v_n$ is a prenecklace if and only if $v_1 \dots v_{n-k+1} \geq v_k \dots v_n$ for all k such that $1 \leq k \leq n$. [3]

The Lyndon words of length d for $d|n$, when extended to length n , form all the necklaces of length n . The Lyndon words of length d for $d \leq n$, when extended to length n , form all the prenecklaces of length n . [4] We describe the relationship between the prenecklaces, necklaces and the Lyndon words in the Table 2 and Figure 2 below for $n = 4$.

Prenecklaces	Lyndon Words	Necklaces
1111	1	1
1110	1110	1110
1101	110	
1100	1100	1100
1010	10	10
1001	100	
1000	1000	1000
0000	0	0

Table 2. Comparison of Type of Words

There is a bijective mapping of the set of all Lyndon words of lengths one through n onto the set of all prenecklaces of length n by taking each of the Lyndon words of length, $d \leq n$ and extending them to length n . These are then the outputs, as defined by Θ , to make the n -long prenecklaces. [4]

Ruskey shows that to count $P_2(n)$, the binary prenecklaces of length n , we need only sum $L_2(i)$ where $1 \leq i \leq n$, all of the Lyndon Words of smaller or equal length:

$$P_2(n) = \sum_{i=1}^n L_2(i). \quad [3]$$

Equation 1

Likewise there is a bijection from the set of all Lyndon words of length d , such that $d|n$, onto the necklaces of length n . To count $N_2(n)$, the necklaces of length n , we need only to sum all of the Lyndon words of length d such that $d|n$:

$$N_2(n) = \sum_{d|n} L_2(d). \quad [5]$$

Equation 2

These relationships imply that the necklaces are a subset of the prenecklaces. This is illustrated in the following figure:

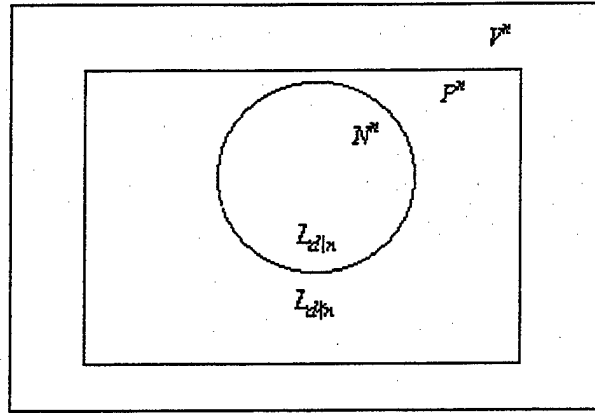


Figure 2. Necklace Algorithm's Reduction of Search Space

Prior to the publication of the Necklace Algorithm [1], generation of the necklaces was accomplished by selecting an n -tuple and performing n cyclic shifts to determine the representative of the necklace class. Graphically, this would be similar to picking a point in the space V^n , and moving from point to point n times until one point was determined that is the representative of the necklace class, N^n . By step 0, the Necklace Algorithm begins the search for necklaces in N^n . By Θ , the search for the next necklace continues in the set of prenecklaces, P^n . [3] By constraining our search to this subset of all n -tuples, we realize a savings in the number of strings that are considered as a possible necklace. Then the J -check removes the need to perform the cyclic rotation to test whether the n -tuple represents the necklace.

Table 3 demonstrates two alternative methods of counting the prenecklaces of length n . First the partition based on the number of Lyndon words of length i , $L_2(i)$, is given. [6] Meanwhile, the Necklace Algorithm partitions the prenecklaces into their respective m classes. Table 3 is provided for the case $n = 11$.

i	$L_2(i)$	m	$ \Theta(11, m) $
-	-	-6	1
1	2	-5	1
2	1	-4	2
3	2	-3	4
4	3	-2	8
5	6	-1	16
6	9	0	32
7	18	1	61
8	30	2	105
9	56	3	128
10	99	4	53
11	186	5	1
$P_2(11)$	412	$ \Theta(11) $	412

Table 3. Comparison of Counting Prenecklaces

A particular prenecklace will appear in different locations in the two tables. (e.g., 1^{11} is located in the rows $i = 1$ and $m = -6$, while 0^{11} appears in row $i = 1$ and $m=5$.) The discussion of how to count the prenecklaces with respect to the class m appears later and is the major contribution of this thesis. By carefully counting the savings in the Necklace Algorithm listing we determine the location of any particular n -tuple in the prefer-ones de

Bruijn sequence. The savings measure for each m class and its difference from the power of 2 described by the respective degrees of freedom is the number of necklaces appearing as enumerated in the second part of Table 3.

4. Necklaces

By step 2, the J -check removes those prenecklaces that are not necklaces. From our statements in the previous section (section 3), step 2 therefore removes the Lyndon words of length $j \nmid n$. This results in additional savings to those observed for step 1. The following theorem verifies the claim that step 2 produces only necklaces:

Theorem 2: A vector, V^n , satisfying step 1 of the Necklace Algorithm, is a necklace if and only if $j \mid n$.

Proof: Given the vector, $V^n = v_1 v_2 \dots v_n$, suppose $n = tj$. then $v_1 v_2 \dots v_n = (v_1 \dots v_{j-1} 0)^t$. Clearly this is a necklace only when $v_1 \dots v_{j-1} 0 > v_i \dots v_{j-1} 0 v_1 \dots v_{i-1} \quad \forall i \neq 0$.

Now let $n = tj + s$ with $1 \leq s < j$, and assume without loss of generality that $v_1 \dots v_{j-1} 0 > v_i \dots v_{j-1} 0 v_1 \dots v_{i-1} \quad \forall i \neq 0$, as before. Then $v_1 \dots v_s v_1 \dots v_{j-1} 0 > v_1 \dots v_{j-1} 0 v_1 \dots v_s$, since dropping the first s bits in each string yields $v_1 \dots v_{j-1} 0 > v_{s+1} \dots v_{j-1} 0 v_1 \dots v_s$ and the prenecklace is not a necklace. \square

Alternative to Equation 2 above, we also have MacMahon's formula for the number $N(n,2)$ of binary necklaces:

$$N(n,2) = \frac{1}{n} \sum_{d|n} \phi(d) 2^{n/d} . \quad [7]$$

Equation 3

Here $\phi(d)$ is Euler's totient function and the summation is over all divisors of the necklace length, n .

As mentioned in the discussion of prenecklaces above, the formula does not partition the necklaces into classes as does the Necklace Algorithm. However, the Necklace Algorithm does not enumerate the necklaces in classes as we would like. (Note: see Appendix A)

We perform some elementary analysis to compute the size of the classes for this parameter. From the definition of the classes, evidently only one necklace class has a 1 in the last position, namely 1^n . Any other prenecklace that ends in a 1 can simply be cyclically rotated to the right to place the 1 in the first position, making the resulting vector bigger than the original vector. e.g., $1101 \rightarrow 1110$ and $1110 > 1101$.

From the previous statements, an elementary upper bound for the number of members in a class can be determined. Since the last position of a necklace must be a 0, an upper-

bound is 2^{DOF-1} . The savings described above shows how far below this power of 2 the number of necklaces actually is. By careful analysis of the Necklace Algorithm, this upper bound can be tightened.

II. COUNTING THE NECKLACE ALGORITHM

A. PRENECKLACE COUNT

1. Table of Data for Missed n-tuples

Our ultimate goal is to better understand the "prefer-ones" de Bruijn sequence. We wish to show where each of the binary n-tuples appears in the sequence. The first step is to determine to which necklace the n-tuple belongs. For each necklace it is possible to determine its class, as the binary n-tuples have been partitioned into classes according to the longest leading substring of ones by the Necklace Algorithm. If we can count how many necklaces appear in the sequence for the first time as members of a necklace class containing no longer strings of ones we will have found the location of the n-tuple. We choose, in fact, to count how many necklaces fail to appear in this way (i.e., those necklaces that appear earlier in the sequence as members of other necklace classes containing longer strings of ones as well). That is, our analysis proceeds as we count the strings that are missing from the m -class of size 2^{l-m+1} (Note: see Table 3 and subsequent discussion).

The computer code used to implement the Necklace Algorithm is modified to count the number of n-tuples

omitted (in the above sense) from inclusion in the respective m class in the list of prenecklaces. For the respective m classes the number of possible n -tuples with $n-(l-m-1)$ degrees of freedom is $2^{n-(l-m-1)}$. Not all of these appear employing the Necklace Algorithm as we have noted. The number of omissions is used as a performance measure for the efficiency of the algorithm as stated above and in Ruskey. [3] The entire table of data is included in Appendix A. These tables in Appendix A, as presented, are segregated by odd and even values for the length of the n -tuples. Our analysis will be for the case in which n is odd. When n is even the analysis is similar. The data in Table 4 and Table 5 is extracted from Appendix A. This data specifies the number of n -tuples missing from the list of prenecklaces of length n in the class m .

	$n = 2l + 1$				
$l-m$	7	9	11	13	15
$l-4$	0	0	0	0	0
$l-1$	3	3	3	3	3
$l-2$	21	23	23	23	23
$l-3$	63	104	128	136	138
$l-4$	--	255	459	628	704
$l-5$	--	--	1023	1930	2871
$l-6$	--	--	--	4095	7926
$l-7$	--	--	--	--	16383

Table 4. Number of Missing n -tuples (Odd)

	$n = 2l$				
$l-m$	6	8	10	12	14
$l-0$	0	0	0	0	0
$l-1$	1	1	1	1	1
$l-2$	9	9	9	9	9
$l-3$	31	48	56	58	58
$l-4$	--	127	221	288	314
$l-5$	--	--	511	946	1353
$l-6$	--	--	--	2047	3920
$l-7$	--	--	--	--	8191

Table 5. Number of Missing n-tuples (Even)

Table 4 and Table 5 provide several insights for our further analysis. First, there are no missing n-tuples when the number of leading ones is greater than or equal to l , that is, when $m < l$. Second, when there are no leading ones, i.e., $m = l$, the only appearing n-tuple is $00\dots 0 = 0^n$. Thus, there are $2^{n-1}-1$ n-tuples missing that have the initial string of one's being zero long. For example for $n = 6$, $l=3$, the last n-tuple from the Necklace Algorithm belonging to the class $m = 3$ is 000000, which has $l-m=l-3=3-3=0$, i.e., the leading number of ones is zero long, the degrees of freedom (DOF) is five and 31 n-tuples are omitted in this class. The dashed lines in the tables indicate that no value is defined for the given parameters as the numbers are not meaningful. A third observation is that the numbers in the table seem to increase to a value and remain at that

value. We call this the steady state number for that m class. The values when steady state has been reached are indicated in bold. This steady state number reflects the maximum savings afforded by performing the Necklace Algorithm for that class. Implicitly, the first sub-problem we face is to count the size of the steady state set of missing n -tuples for a given class m . Further, we show for which value of n the steady state occurs for a given class m .

The notion of a class reaching steady state is demonstrated in Table 6. Consider the following n -tuples from the class $m = 2$, which are missing from the list of prenecklaces for the lengths of n : 7, 9, and 11. We do not include the value of $n = 5$, which for the class $m=2$ has zero leading 1's. Thus there are 15 missing members in the class, namely the non-zero 5-tuples with a leading 0.

<u>$n = 7$</u>		<u>$n = 9$</u>		<u>$n = 11$</u>	<u>Count</u>
1011111	\Rightarrow	11011111		1110111111	1
1011110	\Rightarrow	110111110		11101111110	2
1011101	\Rightarrow	110111101		11101111101	3
1011100	\Rightarrow	110111100		11101111100	4
1011011	\Rightarrow	110111011		11101111011	5
	\Rightarrow	110110111		11101101111	6
1011010	\Rightarrow	110111010		11101111010	7
1011001	\Rightarrow	110111001		11101111001	8
1011000	\Rightarrow	110111000		11101111000	9
1010111	\Rightarrow	110101111		11101011111	10
1010110	\Rightarrow	110101110		11101011110	11
1010011	\Rightarrow	110100111		11101001111	12
1001111	\Rightarrow	110011111		11100111111	13
1001110	\Rightarrow	110011110		11000111110	14
1001101	\Rightarrow	110011101		11100111101	15
	\Rightarrow	110011011		11100111011	16
1001100	\Rightarrow	110011100		11100111100	17
1001011	\Rightarrow	110010111		11100101111	18
1001010	\Rightarrow	110011010		11100111010	19
1000111	\Rightarrow	110001111		11100011111	20
1000110	\Rightarrow	110001110		11100011110	21
1000101	\Rightarrow	110001101		11100011101	22
1000011	\Rightarrow	110000111		11100001111	23
21		23		23	

Table 6. Increasing Missing n -tuples to Steady State

At $n = 7$ the $m = 2$ class is not yet at steady state; $n=9$ is the first value of n that the class $m = 2$ is at steady state. At $n = 11$, the Necklace Algorithm produces a list of necklaces that remains at steady state and there are twenty-three n -tuples omitted for the class $m = 2$ for each succeeding value of n . Comparing the three columns, we see that each missing n -tuple can be mapped (\Rightarrow) to an n -tuple in the successive column by adding a 1 to the leading substring (this is required for to maintain the class

structure), and adding an additional 1 to the longest string of 1's in the trailing substring. Proceeding from $n = 7$ to $n = 9$, there occasionally are two opportunities to add the second 1. It is interesting to note that from $n = 9$ to $n=11$ each missing n -tuple maps to one and only one n -tuple in the next column. For example, 1011011 produces both 110111011 and 11011011. In section 3 we show when such possibilities for two successors end and steady state is therefore achieved.

2. Plot of Missing n -tuples Relative to m

An approximation to the maximum number of missing n -tuples is made by analyzing the data in Appendix A. We extract from the data the first length n for each class m at which steady state is reached. Our analysis includes the values for the parameter n ($2l+1$ for odd and $2l$ for even), l , m , $l-m$, the degrees of freedom and the maximum number of missed n -tuples. The number of n -tuples missed is transformed using the logarithm base two to aid in the scaling on the graph. (Base 2 seems a natural choice since we are concerned with a binary alphabet.)

$2l+1$	l	m	$l-m$	DOF	#missed	$\log_2()$
3	1	1	0	2	3	1.585
9	4	2	2	6	23	4.524
15	7	3	4	10	138	7.109
21	10	4	6	14	740	9.531
27	13	5	8	18	3720	11.861
33	16	6	10	22	17936	14.131

Table 7. Steady state for odd values of n

$2l$	l	m	$l-m$	DOF	#missed	$\log_2()$
6	3	2	1	4	9	3.170
12	6	3	3	8	58	5.858
18	9	4	5	12	324	8.340
24	12	5	7	16	1672	10.707
30	15	7	8	21	8208	13.003
36	18	8	10	25	38944	15.249

Table 8. Steady state for even values of n

There is evidently a strong relationship between l and the number of missing n -tuples. The following graphs of l vs. $\log_2(\text{\#missing } n\text{-tuples})$ illustrates this nearly linear relationship. For reference, a line of the form $y = x$ is plotted in dashed lines as well.

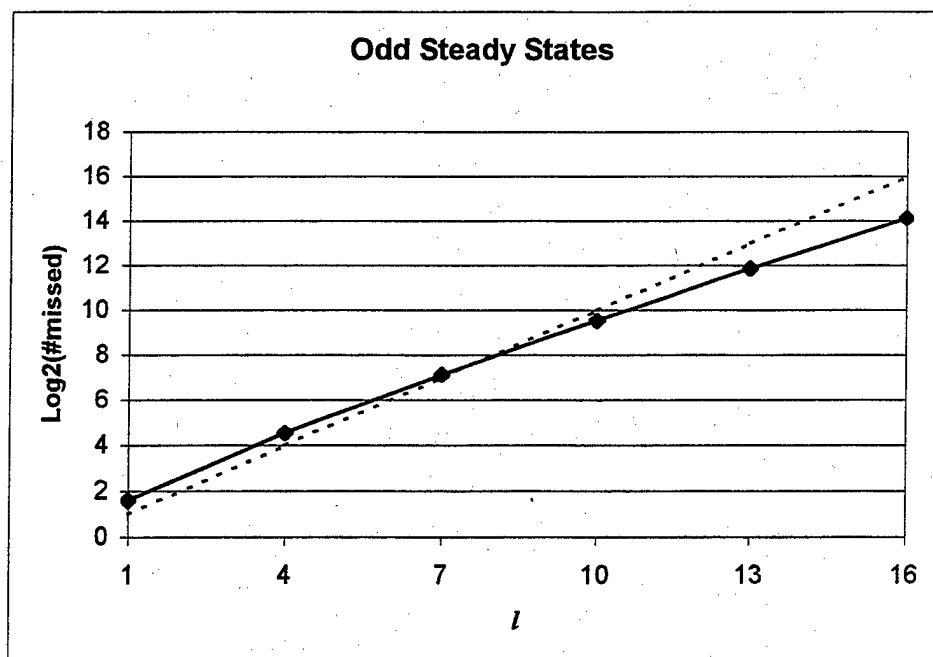


Figure 3. Regression of Steady States (Odd)

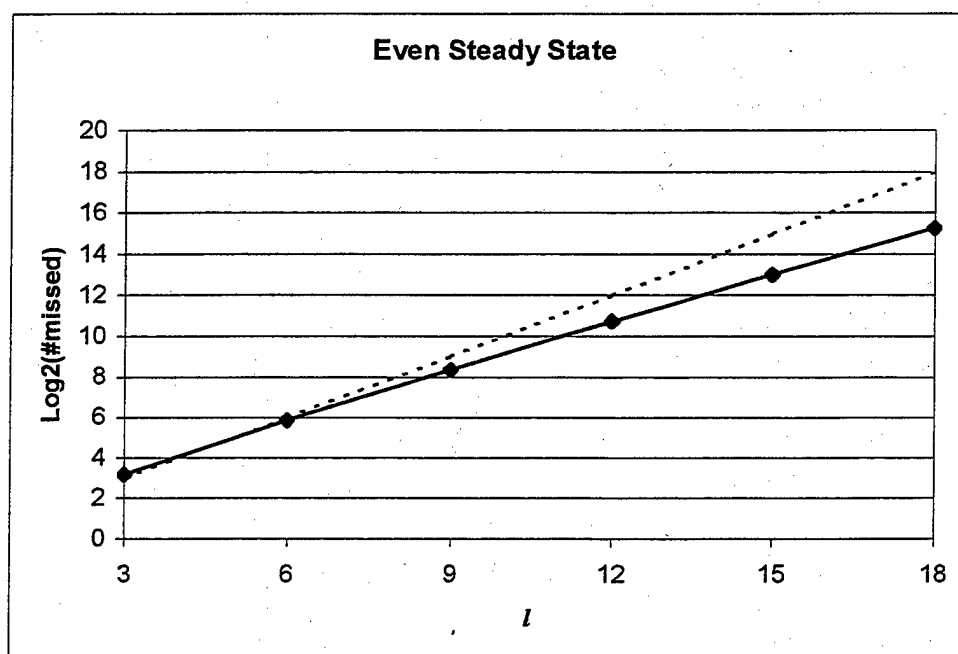


Figure 4. Regression of Steady States (Even)

This linearity is checked with a linear regression of the data where the expected sum of squares yields a correlation of determination (R^2) of 0.99790 for the odd length n -tuples and 0.9989 for the even n -tuples. This leads us to propose $2^l \approx$ steady state value. However, it is also evident that this is only a loose upper bound as l (and therefore n) grows. Our further analysis will also serve to tighten this bound.

3. Finding Steady State

The Necklace Algorithm proceeded using m as a parameter. The above argument establishes l as a determining factor for the number of missing n -tuples (which is also in a linear relationship with n). For the odd values of n , steady state is reached for $n = 3, 9, 15, 21, 27, 33, \dots$. Evidently steady state is reached when $n \equiv 3 \pmod{6}$. We consider the structure of the n -tuples when the steady state is reached. The predecessor and the first n -tuple in the class where steady state is reached exhibit a similar property among the classes.

m	l	$2l+1$	$[l-m+1]$		$\Theta[l-m+1]$
1	1	3	$\Theta(\overbrace{100}^l)$	\Rightarrow	$\overbrace{000}^{l-1}$
2	4	9	$\Theta(\overbrace{111000000}^{l-1})$	\Rightarrow	$\overbrace{110110110}^{l-2}$
3	7	15	$\Theta(\overbrace{111110000000000}^{l-2})$	\Rightarrow	$\overbrace{111101111011110}^{l-3}$
4	10	21	$\Theta(\overbrace{111111100000000000000}^{l-3})$	\Rightarrow	$\overbrace{1111101111110111110}^{l-4}$

Table 9. Examples of First Member of Steady State Class

By the Necklace Algorithm, the last one in an n -tuple is complemented producing $l-m-1$ ones and a zero. It is apparent that the length, $l-m$, of this substring pattern divides n with a cofactor of 3. We also consider larger values for n to ensure that steady state is maintained. Table 10 illustrates the first output of the Necklace Algorithm for the class $m = 2$ for various lengths of n .

n	l	$[l-m+1]$		$\Theta[l-m+1]$
7	3	$\Theta(\overbrace{1100000}^{l-1})$	\Rightarrow	$\overbrace{1010101}^{l-2}$
9	4	$\Theta(\overbrace{111000000}^{l-1})$	\Rightarrow	$\overbrace{110110110}^{l-2}$
11	5	$\Theta(\overbrace{11110000000}^{l-1})$	\Rightarrow	$\overbrace{11101110111}^{l-2}$
13	6	$\Theta(\overbrace{1111100000000}^{l-1})$	\Rightarrow	$\overbrace{1111011110111}^{l-2}$

Table 10. Examples of First Member of Class $m = 2$

It can be confirmed from the data in Table 4, for the class $l=2$, steady state is reached at $2l+1 = 9$, and continues for all larger values of l . The first output of the Necklace Algorithm for the class $m = 2$, and for the value of n for which steady state is reached has the following form (R is used to abbreviate the remainder):

$$\overbrace{1\dots 1}^{l-2} \overbrace{01\dots 01}^{l-2} \overbrace{101\dots 101}^R \alpha_n.$$

The term remainder indicates that part of the leading substring that is not completely copied. Evidently, this shows that if l is large enough to allow the leading substring to be copied two times and leave a remainder of length three or greater then steady state will be reached for the class $m = 2$. The classes of larger values of m also exhibit a distinct relationship with l and m . For $n=2l+1=31$, and the classes of $m = 1, 2, \dots, 6$ the following similar characteristics appear (see Table 11).

m	l	Last output class $l-1$		First output class $l-2$	Length of Remainder
1	15	$\Theta(\overbrace{1\dots 100000}^l)$	\Rightarrow	$\overbrace{1\dots 101}^{l-1}\overbrace{\dots 101}^{l-1}$	1
2	15	$\Theta(\overbrace{1\dots 100000}^{l-1})$	\Rightarrow	$\overbrace{1\dots 101}^{l-2}\overbrace{\dots 101}^{l-2}11$	3
3	15	$\Theta(\overbrace{1\dots 100000}^{l-2})$	\Rightarrow	$\overbrace{1\dots 101}^{l-3}\overbrace{\dots 101}^{l-3}1111$	5
4	15	$\Theta(\overbrace{1\dots 100000}^{l-3})$	\Rightarrow	$\overbrace{1\dots 101}^{l-4}\overbrace{\dots 101}^{l-4}111111$	7
5	15	$\Theta(\overbrace{1\dots 100000}^{l-4})$	\Rightarrow	$\overbrace{1\dots 101}^{l-5}\overbrace{\dots 101}^{l-5}1111111$	9
6	15	$\Theta(\overbrace{1\dots 100000}^{l-5})$	\Rightarrow	$\overbrace{1\dots 101}^{l-6}\overbrace{\dots 101}^{l-6}\overbrace{\dots 11}^{l-6}$	1

Table 11. Examples of First Output for Classes ($l = 15$)

(Note: for $m = 6$, steady state has not been reached for $n = 31$.)

This suggests the following theorem:

Theorem 3:

Steady state for the Θ -step of The Necklace Algorithm will be reached for the class of m at $n = 2l+1$, when $l+2 \geq 3m$.

Proof: From Table 10, we see that the substring "110" is copied three times for $2l + 1 = 9$. For larger n and the same m , the initial string $1^{l-m}0$ will only be copied twice plus the first $2m-1$ bits. This establishes the following inequality:

$$3(l-m+1) \geq 2l+1=n$$

$$3l-3m+3 \geq 2l+1=n$$

$$l+2 \geq 3m$$

To illustrate Theorem 3, we verify the result for the n-tuple of length 31, showing that the class of $m = 6$ and the number of leading ones ($l-m = 15-6$) is not at steady state:

m	l	$l+2$	$3m$
1	15	17	3
2	15	17	6
3	15	17	9
4	15	17	12
5	15	17	15
6	15	17	18

Table 12. Verification of Theorem 3

(Note: Steady state is achieved at $2l+1 = 33$.)

By the theorem, steady state is obtainable. Because this is not *a priori* apparent we note this specifically in the following corollaries.

Corollary 1:

For a given class, m , the Necklace Algorithm will reach steady state for some $l \geq L$.

Proof: m is a fixed quantity. Suppose that $l+2 < 3m$.

Let $L = 3m$. Increase l such that $l > L$. \square .

Corollary 2:

Steady State is reached for class m if and only if:

$$n/(l-m+1) = 2 \text{ with a remainder of } 2m-1.$$

It is helpful to consider an example of Corollary 2. Let $l = 15$, then $n = 31$. From Table 12, we can compare the results for the classes $m = 1, 2, 3, 4, 5, 6$ using the corollary.

m	$2m-1$	$\left\lfloor \frac{n}{l-m+1} \right\rfloor$	$(l-m+1) \bmod (n)$
1	1	2	1
2	3	2	3
3	5	2	5
4	7	2	7
5	9	2	9
6	11	3	1

Table 13. Compare Results of m and Steady State

(Note: Steady state is reached for all cases but $m = 6$.)

B. NECKLACE COUNT

1. Table of Data for Omitted Necklaces and n-tuples

For the "savings" realized by the J -check, we first examine the difference between the prenecklaces and the necklaces of length n . Equation 1 and Equation 2 respectively compute the values of P^n and N^n , respectively. These values are presented in Table 14.

n	P^n	N^n	$P^n - N^n$
1	2	2	0
2	3	3	0
3	5	4	1
4	8	6	2
5	14	8	6
6	23	14	9
7	41	20	21
8	71	36	35
9	127	60	67

Table 14. Prenecklaces, Necklaces, and the Difference

The number P^n is approximately twice the number N^n . But more information can be obtained by comparing the number of prenecklaces and necklaces for each of the m -classes for a fixed n . In particular, steady state becomes apparent again. Table 15 gives values partitioned by m -class for the case $n = 31$.

m	$P(m)$	$N(m)$	$P(m)-N(m)$
-16	1	1	0
-15	1	1	0
-14	2	1	1
-13	4	2	2
-12	8	4	4
-11	16	8	8
-10	32	16	16
-9	64	32	32
-8	128	64	64
-7	256	128	128
-6	512	256	256
-5	1024	512	512
-4	2048	1024	1024
-3	4096	2048	2048
-2	8192	4096	4096
-1	16384	8192	8192
0	32768	16384	16384
1	65533	32766	32767
2	131049	65522	65527
3	262006	130992	131014
4	523548	261728	261820
5	1044856	522240	522616
6	2079218	1038850	1040368
7	4110418	2052174	2058244
8	8005876	3990960	4014916
9	15076085	7492056	7584029
10	26279377	12974378	13304999
11	38218348	18604412	19613936
12	35857846	16913378	18944468
13	11532736	5064334	6468402
14	269684	97108	172576
15	1	1	0

Table 15. Count of Step Outputs ($n = 31$)

It is apparent that $P(m)-N(m)$ grows by a factor of 2 until $m \geq 1$. Then when the Necklace Algorithm is performed, a necklace representative is found without performing the n rotations and comparisons. But the cost of the algorithm is still about twice the number of necklaces found, since the

prenecklaces have to be determined first. Reviewing Figure 2, the number of prenecklaces that are not necklaces, $P(n) - N(n)$, is evidently on the order of the number of necklaces [1]. Thus the work to determine the necklaces is cut down from the $n2^n$ rotations and comparisons to $O(N(n)) = O(2^n/n)$ as a function of utilizing the Lyndon words.

C. DEFINING STEADY STATE

Steady state has been shown to occur for particular m -classes of prenecklaces in Theorem 3. In the previous section, the prenecklaces are determined to be n -long extensions of the Lyndon words of length d , for $1 \leq d \leq n$. Similarly, the necklaces are determined by extending to length n the Lyndon words of length d , where $d|n$. We now consider partitioning the Lyndon words into m -classes to verify this notion of steady state.

From the partitioning into m -classes of the prenecklaces for a given n , we can count the number of prenecklaces for a given length of n using Equation 2. Since the prenecklaces are d -long Lyndon words extended to length n , we can count the number of prenecklaces in a given class by counting the number of d -long Lyndon words that

begin with the substring $1^{l-m}0$. Thus we can enumerate the Lyndon words for each m -class and for any value of n . Table 16 lists the number of prenecklaces of length nine determined by Lyndon words of length $d \leq 9$ and partitioned by the number $l-m$ of leading 1's:

d	$l-m$									
	9	8	7	6	5	4	3	2	1	0
1	1	0	0	0	0	0	0	0	0	1
2	0	0	0	0	0	0	0	0	1	0
3	0	0	0	0	0	0	0	1	1	0
4	0	0	0	0	0	0	1	1	1	0
5	0	0	0	0	0	1	1	2	2	0
6	0	0	0	0	1	1	2	3	2	0
7	0	0	0	1	1	2	4	6	4	0
8	0	0	1	1	2	4	7	10	5	0
9	0	1	1	2	4	8	14	18	8	0

Table 16. Partitioning Lyndon Words

(Note: The decreasing values for $l-m$ represent the lexicographic order imposed by the Necklace Algorithm. The Necklace Algorithm proceeds by columns, from left to right in the table, producing each n -long extended Lyndon word (prenecklace).

From the table it is evident that the number of prenecklaces in the class $m = 2$ is $(1+1+2+3+6+10+18 =)$ 41. In Table 4, steady state for the class $m = 2$ is 23 missing n -tuples. We verify that $2^{DOF} - P(n) = 64 - 41 = 23$. The

reader can easily check that steady state is reached for the class $m = 1$ (i.e., missing 3 n-tuples in the column $l-m=3$).

Not only can it be determined how many prenecklaces are missing for a particular class, but it can be determined how many are missing for a particular value of d . By precalculating the size of the set of missing n-tuples at steady state for each m -class, it is possible to precisely determine the position for a given necklace. A necklace in a class for which steady state has been achieved can easily be calculated using the known value for a precise position. For other classes, not at steady state, this information can be helpful in establishing a tight upperbound for the number of necklaces in a given class before initiating a sequential search in the list of necklaces.

THIS PAGE INTENTIONALLY LEFT BLANK

III. CONCLUSIONS AND SUGGESTIONS FOR ADDITIONAL RESEARCH

The Necklace Algorithm has been shown to be significantly more efficient than a naive algorithm producing all of the necklaces of a given length in lexicographic order. The algorithm also has been shown to produce all of the Lyndon words of lengths one through n in lexicographic order as prenecklaces. By developing the notion of steady state, it is shown to be possible to find the exact position of the necklace representative of any particular n -tuple given that the leading substring has at least l ones. If the representative necklace has fewer leading 1's, a table is available to count the missing necklaces for the given class m . The class must be at steady state to ensure that the table size is feasible. The condition for steady state is shown to be $l+2 < 3m$. Additional methods must be employed when steady state has not been achieved.

To demonstrate the application of our results, an example is given. We choose an n -tuple:

$$V^S = 000110111110101.$$

First, we determine the n -tuple's parameters using Definition 1. The longest substring of 1's is 11111; and

since $l = 7$ we find $m = 2$. Now, we check for the steady state condition:

$$7+2 \geq 3(2) \rightarrow 9 > 6 \Rightarrow \text{Class } m = 2 \text{ is at steady state.}$$

From Appendix A, the steady state values for odd values of n are 3, 23, 138 in the classes $m = 1, 2$, and 3. The representative of the necklace class is: 111110101000110. This representative has no internal periodicity so the J -check would have shown that this n -tuple is a Lyndon word of length $d = 15$. There are 242 Lyndon words of length 15 and with $l-m = 5$. To locate the exact necklace then we would enter the string $[l-m-1]$ ($= 111111000000000$). We then apply the Necklace Algorithm and count how many necklaces are produced, until the correct one is found.

From this example and our discussion in the previous section, it is clear that a better understanding of the Lyndon words and the development of an efficient counting technique for a partitioning by class would result in an improved solution for this version of the distance problem. With the knowledge of the location of a necklace relative to an initial position in the "prefer-ones" de Bruijn sequence a distance can be determined [5] and refined quickly in such a manner that it would be computationally efficient. [8] This could result in the development of a new cryptographic

scheme. Our methods and results may also have application in data compression. Certainly, it can be said the solution to this problem is valuable for both academic and practical reasons.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. TABLE OF MISSING N-TUPLES COUNTS

	7	9	11	13	15	17	19	21	23	25	27	29	31
L-0	0	0	0	0	0	0	0	0	0	0	0	0	0
L-1	3	3	3	3	3	3	3	3	3	3	3	3	3
L-2	21	23	23	23	23	23	23	23	23	23	23	23	23
L-3	63	104	128	136	138	138	138	138	138	138	138	138	138
L-4		255	459	628	704	730	738	740	740	740	740	740	740
L-5			1023	1930	2871	3392	3606	3684	3710	3718	3720	3720	3720
L-6				4095	7926	12584	15631	17038	17606	17822	17900	17926	17934
L-7					16383	32157	53717	69872	78079	81646	83100	83670	83886
L-8						65535	129653	225253	305619	349970	370379	379118	382732
L-9							262143	520955	932995	1315402	1542563	1652306	1701131
L-10								1048575	2089242	3830876	5592057	6710766	7275055
L-11									4194303	8369683	15630420	23543815	28890516
L-12										16777215	33508823	63477929	98359882
L-13											67108863	134107119	256902720
L-14												268435455	536601228
L-15													1073741823

Table 17. Missing n-tuples (n odd)

	7	9	11	13	15	17	19	21	23	25	27	29	31
L-0		0	0	0	0	0	0	0	0	0	0	0	
L-1		0	0	0	0	0	0	0	0	0	0	0	0
L-2		2	0	0	0	0	0	0	0	0	0	0	0
L-3		41	24	8	2	0	0	0	0	0	0	0	0
L-4			204	169	76	26	8	2	0	0	0	0	0
L-5				907	941	521	214	78	26	8	2	0	0
L-6					3831	4658	3047	1407	568	216	78	26	8
L-7						15774	21560	16155	8207	3567	1454	570	216
L-8							64118	95600	80366	44351	20409	8739	3614
L-9								258812	412040	382407	227161	109743	48825
L-10									1040667	1741634	1761181	1118709	564289
L-11										4175380	7260737	7913395	5346701
L-12											16731608	29969106	34881953
L-13												66998256	122795601
L-14													268165773
L-15													

Table 18. Forward Difference (n odd)

	6	8	10	12	14	16	18	20	22	24	26	28	30
l-0	0	0	0	0	0	0	0	0	0	0	0	0	0
l-1	1	1	1	1	1	1	1	1	1	1	1	1	1
l-2	9	9	9	9	9	9	9	9	9	9	9	9	9
l-3	31	48	56	58	58	58	58	58	58	58	58	58	58
l-4		127	221	288	314	322	324	324	324	324	324	324	324
l-5			511	946	1353	1558	1636	1662	1670	1672	1672	1672	1672
l-6				2047	3920	6038	7319	7878	8094	8172	8198	8206	8208
l-7					8191	15983	26075	33160	36599	38044	38614	38830	38908
l-8						32767	64609	110209	146483	165716	174327	177932	179388
l-9							131071	259975	459035	635182	736127	783764	804579
l-10								524287	1043452	1892322	2715891	3222158	3472217
l-11									2097151	4182089	7743350	11486697	13940496
l-12										8388607	16747871	31514779	48164396
l-13											33554431	67037905	127749326
l-14												134217727	268262880
l-15													536870911

Table 19. Missing n-tuples (n even)

	6	8	10	12	14	16	18	20	22	24	26	28	30
l-0		0	0	0	0	0	0	0	0	0	0	0	
l-1		0	0	0	0	0	0	0	0	0	0	0	0
l-2		0	0	0	0	0	0	0	0	0	0	0	0
l-3		17	8	2	0	0	0	0	0	0	0	0	0
l-4			94	67	26	8	2	0	0	0	0	0	0
l-5				435	407	205	78	26	8	2	0	0	0
l-6					1873	2118	1281	559	216	78	26	8	2
l-7						7792	10092	7085	3439	1445	570	216	78
l-8							31842	45600	36274	19233	8611	3605	1456
l-9								128904	199060	176147	100945	47637	20815
l-10									519165	848870	823569	506267	250059
l-11										2084938	3561261	3743347	2453799
l-12											8359264	14766908	16649617
l-13												33483474	60711421
l-14													134045153
l-15													

Table 20. Forward Difference (n even)

APPENDIX B. CODE FOR NECKLACE ALGORITHM

The following code is a Java program that implements the Necklace Algorithm:

```
import java.util.*;
import java.lang.*;
import java.io.*;
public class Theta {
    public static void main (String args[]) {
        int length = 31;
        int k = 1;
        int sequence [] = new int[length];
        int j = length-1;
        int counter = 0;
        int numNeck = 0;
        int numType = 0;
        int flag = 1;
        int group [] = new int[length+1];
        int sum = 0;
        System.out.println("Finding all binary necklaces of length "+
            length + ".");
        // Sets the necklace of all 1's.
        for (int a = 0; a < length; a++) {
            sequence[a]=k;
            System.out.print(sequence[a]);
        }
        System.out.println(" ");
        // Find last non zero element and decrement it
        while (sequence[0] != 0) {
            j = length-1;
            while (sequence[j] == 0){
                j=j-1;
            }
            // Change last one-bit and fill register.
            sequence[j] = sequence[j]-1;
            if (j != length-1) {
                for (int i = counter+1; i < length-j; i++){
                    sequence[j+i] = sequence[i-1];
                }
            }
            if (length %(j+1) == 0) {
                for (int a = counter; a < length; a++) {
                    System.out.print(sequence[a]);
                }
            }
            counter++;
        }
    }
}
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. TABLE OF LYNDON WORD COUNTED BY PARTITIONS

<i>d</i>	<i>l-m</i>													
	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	1
3	0	0	0	0	0	0	0	0	0	0	0	0	1	1
4	0	0	0	0	0	0	0	0	0	0	0	1	1	1
5	0	0	0	0	0	0	0	0	0	0	1	1	2	2
6	0	0	0	0	0	0	0	0	0	1	1	2	3	2
7	0	0	0	0	0	0	0	0	1	1	2	4	6	4
8	0	0	0	0	0	0	0	1	1	2	4	7	10	5
9	0	0	0	0	0	0	1	1	2	4	8	14	18	8
10	0	0	0	0	0	1	1	2	4	8	15	26	31	11
11	0	0	0	0	1	1	2	4	8	16	30	50	56	18
12	0	0	0	1	1	2	4	8	16	31	58	93	96	25
13	0	0	1	1	2	4	8	16	32	62	114	178	172	40
14	0	1	1	2	4	8	16	32	63	122	221	334	299	58
15	1	1	2	4	8	16	32	64	126	242	432	634	530	90
16	1	2	4	8	16	32	64	127	250	477	840	1194	929	135
17	2	4	8	16	32	64	128	254	498	944	1640	2262	1646	210
18	4	8	16	32	64	128	255	506	989	1862	3190	4265	2893	316
19	8	16	32	64	128	256	510	1010	1968	3682	6222	8072	5126	492
20	16	32	64	128	256	511	1018	2013	3910	7268	12112	15239	9044	750
21	32	64	128	256	512	1022	2034	4016	7776	14362	23608	28824	16028	1164
22	64	128	256	512	1023	2042	4061	8006	15454	28358	45975	54461	28362	1791
23	128	256	512	1024	2046	4082	8112	15968	30728	56024	89592	103008	50328	2786
24	256	512	1024	2047	4090	8157	16198	31836	61074	110634	174507	194725	89249	4305
25	512	1024	2048	4094	8178	16304	32352	63490	121422	218542	340034	368356	158598	6710
26	1024	2048	4095	8186	16349	32582	64604	126592	241352	431607	662419	696663	281830	10420
27	2048	4096	8190	16370	32688	65120	129024	252442	479802	852524	1290716	1318138	501538	16264
28	4096	8191	16378	32733	65350	130140	257658	503358	953744	1683767	2514709	2493968	892857	25350
29	8192	16382	32754	65456	130656	260096	514568	1003736	1895966	3325762	4900000	4720098	1591282	39650
30	16383	32762	65501	130886	261212	519800	1027594	2001434	3768843	6568697	9547486	8933851	2837467	61967
31	32766	65522	130992	261728	522240	1038850	2052174	3990960	7492056	1.30E+07	1.68E+07	1.68E+07	5064334	97108

Table 21. *n* = 31 (part I)

	<i>l-m</i>																
<i>d</i>	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	2
19	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	2	4
20	0	0	0	0	0	0	0	0	0	0	0	0	1	1	2	4	8
21	0	0	0	0	0	0	0	0	0	0	0	1	1	2	4	8	16
22	0	0	0	0	0	0	0	0	0	0	1	1	2	4	8	16	32
23	0	0	0	0	0	0	0	0	0	1	1	2	4	8	16	32	64
24	0	0	0	0	0	0	0	0	1	1	2	4	8	16	32	64	128
25	0	0	0	0	0	0	0	1	1	2	4	8	16	32	64	128	256
26	0	0	0	0	0	0	1	1	2	4	8	16	32	64	128	256	512
27	0	0	0	0	0	1	1	2	4	8	16	32	64	128	256	512	1024
28	0	0	0	0	1	1	2	4	8	16	32	64	128	256	512	1024	2048
29	0	0	0	1	1	2	4	8	16	32	64	128	256	512	1024	2048	4096
30	0	0	1	1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192
31	0	1	1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384

Table 22 $n = 31$ (part II)

LIST OF REFERENCES

1. Fredricksen, H. and Kessler, I., "An Algorithm For Generating Necklaces Of Beads In Two Colors," *Discrete Mathematics*, 61, pp. 181-188, 1986.
2. Fredricksen, H. M., Maiorana, J. "Necklaces of Beads in k Colors and k-ary de Bruijn Sequences," *Discrete Mathematics*, V. 23, #3, 1978
3. Ruskey, F., Generating Necklaces, *J. Algorithms*, 13(1992) 414-430.
4. Cattell, K, Ruskey, F., Sawada, J., Miers, C., Serra, M., Generating Unlabeled Necklaces and Irreducible Polynomial over GF(2), to be published.
5. Golomb, S. W. Shift Register Sequences, Holden-Day, Inc. 1967
6. Lothaire, M. Encyclopedia Of Mathematics and Its Applications, Volume 17, Combinatorics On Words, pp. 8-9, Addison Wesley Publishing Company, Inc., 1983.
7. Graham, R.L., Knuth, D. E., and Patashnik, O., Concrete Mathematics: A Foundation For Computer Science, 2d ed., Addison-Wesley Publishing Company, Inc. 1994
8. Mitchell, C. J., Etzion, T., Paterson, K. G., "A Method for Constructing Decodable de Bruijn Sequences," *IEEE Transactions Information Theory*, IT-42 (1996), 1472-1478

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

Defense Technical Information Center	2
8725 John J. Kingman Rd. STE 0944	
Ft. Belvoir, Virginia 22060-6218	
Dudley Knox Library.....	2
Naval Postgraduate School	
411 Dyer Rd.	
Monterey, California 93943-5101	
Naval Postgraduate School	5
ATTN: Professor Harold M. Fredricksen	
1411 Cunningham Rd.	
Monterey, California 93943-5216	
Captain Douglas M. Matty.....	5
2 University Circle	
Box 1492	
Monterey, California 93943-5101	
Department of Mathematics.....	1
West Point, New York 10996	